APPLICATION PERFORMANCE MANAGEMENT BEST PRACTICES

2013 State of the Art and Trends

A collection of articles from the Compuware APM Center of Excellence

COMPUWARE APM
DYNATRACE | GOMEZ

Compuware
THE TECHNOLOGY PERFORMANCE COMPANY

WELCOME

I am proud to present this collection of articles on APM best practices and trends written by the Compuware Center of Excellence. The Center of Excellence is a global group of Performance Engineers dedicated to advancing the state of the art in Application Performance Management. The following articles were hand-picked to provide insight into the key application performance issues and challenges organizations will face in 2013.

The articles are organized around the following five key trends that will impact how applications are managed and optimized in 2013:

1. **Complexity Will Continue to Accelerate Around Management of Business Critical Apps.** IT environments continue to become more complex—at the edge of the internet, in the cloud and in the data center—increasing the need for new generation APM with unified, real-time insight across the application delivery chain, including third-party and cloud services.

2. **Performance Analytics Will Be a Requirement For a Complete APM Strategy.** Organizations now collect unprecedented volumes of data. The key to making sense of this will be systems that automate analytics and deliver actionable insight and answers, not just more data.

3. **The Mobile Surge Will Drive the Industrialization of Mobile Applications.** Users now access applications from an unprecedented number of mobile devices. Mobile applications are fast becoming the primary engagement model and mobile end users expect the same performance as from desktops.

4. **APM Will Become a Strategic Component of Big Data.** Big data is trending from experimental projects to becoming an enterprise-wide analytics platform. Expanding data volume, variety, velocity, and complexity necessitates a new approach to enterprise analytics. Integrating new generation APM into big data environments will become a best practice for organizations to eliminate risks and costs associated with poor performance, availability, and scalability.

5. **A Lifecycle Approach to APM Will Drive Adoption of DevOps and Agile Operations.** Leading organizations are already using new generation APM to support unified management of the application lifecycle—development, testing, and production. With DevOps and Agile gaining ground, more organizations will take a lifecycle approach to APM.

I would like to thank all the authors who contributed to this publication, and to you for your continued readership. You can find our latest and archived articles at **http://apmblog.compuware.com**.

Andreas Grabner

Compuware APM Center of Excellence.

2013 State of the Art & Trends

THIRD-PARTY ISSUES AND THE PERFORMANCE RIPPLE EFFECT

— Stephen Pierzchala, September 12, 2012

On September 10, 2012, Go Daddy, a major Internet registry and DNS provider for millions of domain names experienced an outage that caused a substantial impact for companies and people around the globe while major online sites in the US experienced no or minimal performance impact from this event.

How can this be? How can a major online name registry and DNS provider go offline for 4–5 hours and only have an indirect or minimal impact on some of the largest online companies in the US?

For large online companies, control and management of DNS is seen as a critical part of their infrastructure and they either maintain direct control of their DNS namespace or contract its management out to well-known third parties such as content-delivery networks (CDNs) and specialized global DNS providers. As a result, the primary domain names for these firms were less likely to be affected by the loss of DNS servers.

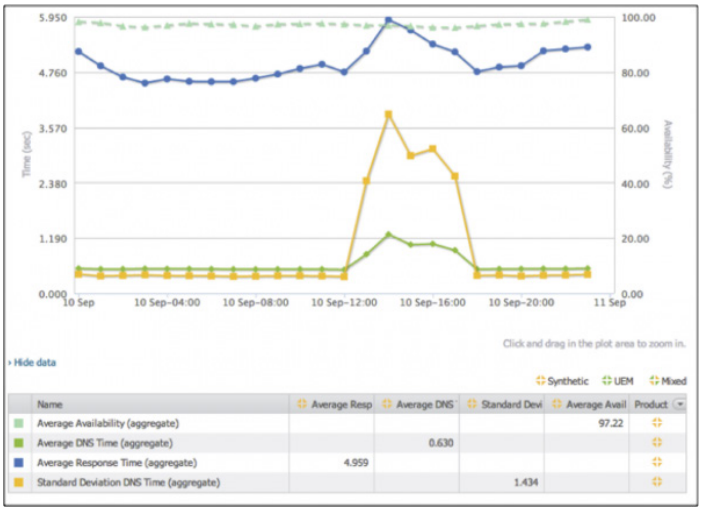
Small businesses, groups, and individuals who do not have the resources to either hire people or contract with firms to take over their DNS management rely on the domain registries or hosting providers they have selected to provide this service for them. For a firm that is doing business or delivering third-party services to other to companies, losing DNS service is the same as being thrown off the Internet, if only for a short time.

The outage at Go Daddy rippled across the Internet and had an impact on thousands (potentially millions) of individuals, small businesses, and third-party service providers.

The protection that many of the top online businesses built into their infrastructure to prevent such a catastrophe from affecting them didn't leave their applications completely unaffected. The interconnected nature of modern online applications, with the increasing reliance on third-party service providers from CDNs to customer-service surveys means that protecting your DNS doesn't protect you from an outage that someone else suffers.

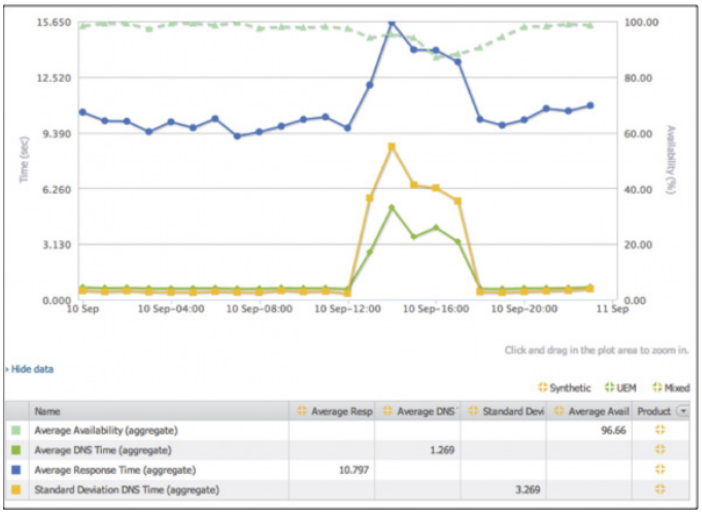
Data collected for the Gomez US Retail Product Order, Retail Homepage, and Top Traffic benchmarks shows three key trends during the outage:

- Overall aggregated response times were elevated.
- Aggregated DNS lookup times increased by 3x to 4x.
- Variability of DNS results increased from less than 0.4 seconds to nearly 4.0 seconds at the peak of the issue.
- But not all firms were affected equally. Some firms saw a much larger effect on performance as tracked by the same metrics. In the following graph, isolating the aggregated metrics for some of the most affected firms shows that availability, response times, DNS times, and DNS variability were substantially higher than the entire population of benchmarks.



Response time, availability, and DNS metrics for September 10, 2012

The selection of third-party providers appears to have been critical in this instance. Firms that used vendors that relied on the affected DNS provider suffered a far greater performance effect than those who used other firms delivering the same service. This points out that there are new factors that companies need to consider when selecting third parties and when companies delivering these third-party services design their infrastructure.



Most affected firms—DNS outage—September 10, 2012

The two primary takeaways from this event are as follows:

- Understand the true impact of third-party content on performance and user experience. Knowing whether a third-party service outage will be an annoying distraction or whether it could shut down the application, with a matching loss in revenue and brand value, is critical for an organization.
- A redundant and scalable infrastructure design is a critical competitive differentiator when selecting third-party content providers. Third-party services that can clearly demonstrate that steps have been taken to prevent their services from causing performance degradation or application outages will show that they want to be partners in the success of their customers' businesses.

The Go Daddy outage clearly highlighted that the failure of a critical Internet resource can ripple outward, affecting even those firms would, at first glance, seem to have taken steps to protect themselves from just such an occurrence.

PARADIGM SHIFT IN COST STRUCTURE AND ITS RISK IN PUBLIC CLOUD ENVIRONMENTS

— Daniel Kaar, May 29, 2012

Today, many decision-makers tend to favor public cloud environments over their own data centers. Often the reason is easier and quicker scaling to cover daily or seasonal peaks, and it sounds promising as a cheaper storage solution. However, it's just a different means of paying the bill, which is at first glance financially attractive. If we have a closer look, it turns out to be quite an issue, especially for financially responsible folks: it becomes more complex and less predictable. Let's have a look at the paradigm shift and how the current approach in enterprise data centers compares to running your business in the public cloud:

THE ENTERPRISE DATA CENTER

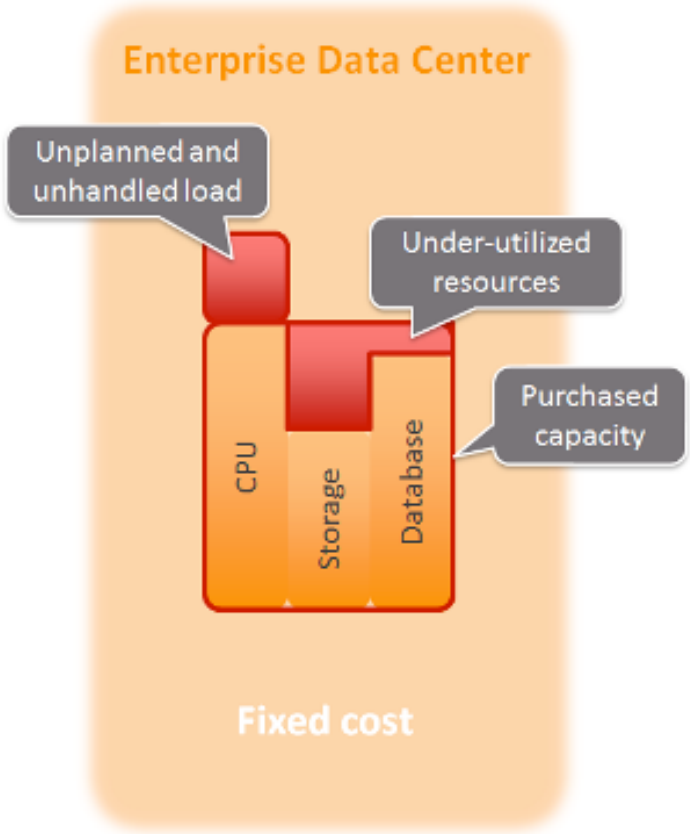
We are used to large, planned infrastructure investments that we made with the expectation that it be capable of handling the anticipated load. From a financial perspective, dealing with a high initial cost that is paid up front isn't quite ideal. Finance folks came up with approaches like leasing and renting to address exactly that pain. However, I don't want to go into further financial details here; rather, I'd like to talk about the planning process, which may look roughly like the following:

1. Estimate the load on the application for the upcoming period.
2. Determine the peak load at given a time.
3. Assess resources (hardware) needed to handle the load.
4. Decide whether to buy (or rent, lease, etc.) additional resources (hardware).

What we are actually doing here is capacity planning. More precisely, it is capacity planning of a rather inertial system—it will take quite a while to have the hardware delivered and up and running, and thus, careful planning is key. Still, we are facing all the familiar issues of capacity planning: if we plan for insufficient hardware, we may be unable to cover the load and might face one or more of the following issues:

- Slow application
- Unsatisfied users
- Loss of revenue

With an enterprise data center, if we aren't capable of accurately assessing our upcoming demand, we have to deal with either costly unutilized resources and over-provisioning or the risk of being unable to serve possible customers. Financially, we have the risk of making unprofitable investments, but the advantage of having little chance of unpredictable costs.



Cost structure of an enterprise data center

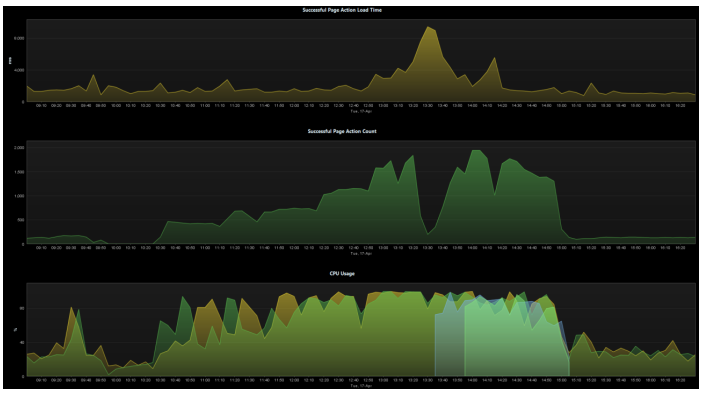
THE PUBLIC CLOUD

When looking at public cloud environments like Amazon EC2 or Windows Azure, financial folks will be pleased since cost and benefit possible to correlate within a short time period. However, most likely this isn't our primary goal since we could have done this years ago by accepting outsourcing (or similar) offers. Rather, we move or develop our applications based on such cloud environments to address the consequences of the inaccurate planning we discussed above:

- We want to eliminate the high cost of over-provisioning—i.e., to buy a lot of infrastructure for covering peak scenarios and growing demand.
- We want our application to scale easily in case of (unpredicted) increasing load.

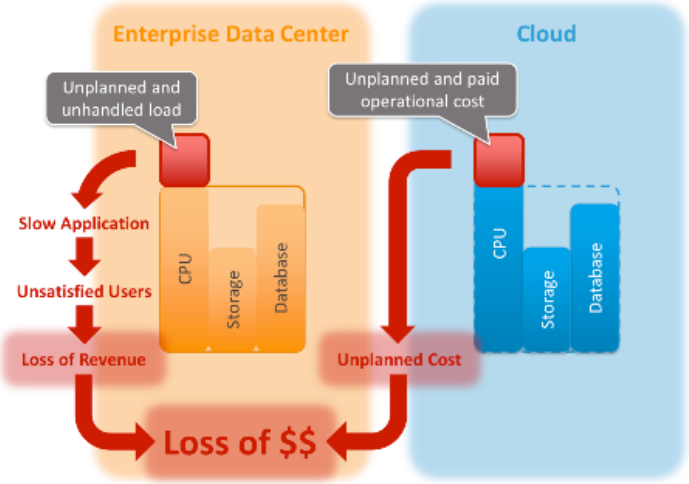
The solution to both of these points is on-demand provisioning, which is brought to perfection in public cloud offerings. With this, we turn the data center issues into the following consequences:

- Stable response times and user experience (see the image below)
- Low chance of revenue loss
- Potential increase in operational cost



On-demand provisioning: Load increases (middle chart) and response time increases (top chart) when both CPUs reach their capacity (bottom chart). After adding a third and fourth CPU (bottom chart) we maintain the load (middle chart) with our desired load time (top chart).

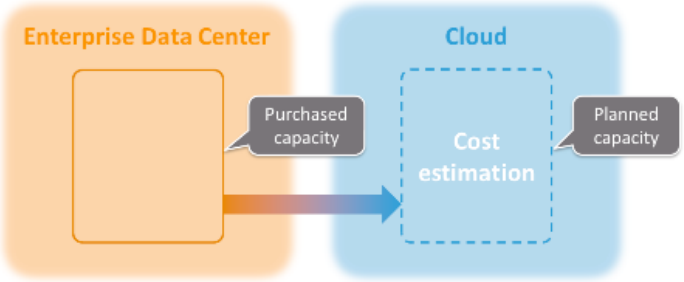
Although stable response times and low chance of revenue loss seem promising for having solved the problems, the possible increase in operational cost brings a new issue to the table. Since this increase can be completely unplanned and its extent is unknown, we may face uncalculated operational costs, leading to the same result as in the data center: at the end of the day we do not earn money. In the data center, we had insufficient resources to cover the load and earn the money, and in the cloud we have a higher production cost to cover the load, which can eat up the earnings.



The cloud risk shift: We lose money in both cases—in the data center we could not earn the money, but in the cloud we have too-high production costs.

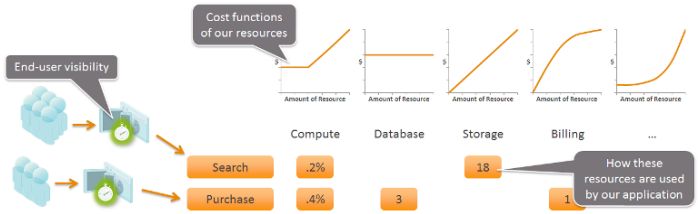
Therefore, we still have to do some kind of capacity planning when operating our applications in cloud environments—but driven from a new origin: since we scale just in time, we actually don't need to plan to handle our traffic; rather, we plan to become aware of the cost for the upcoming period. (Of course, this is true only to a certain extent—if we need a vast amount of compute hours, for instance, because we are expecting a huge seasonal peak, we would file a reservation in advance that still is driven by planning to handle traffic.) However, planning for public cloud environments could look like this:

- Estimate the load on the application for the upcoming period.
- Assess resources.
- Calculate cost for resources.



In public clouds, we plan not to cover load, but rather to estimate cost.

And here is the bad news: this is going to be quite complex. Public clouds offer a wide range of resources; most of them differ in their cost function. On top of that, our application obviously uses different resources to varying extents (resource usage). And to make matters even worse, all this depends on how our users interact with our application—which is hardly plannable.



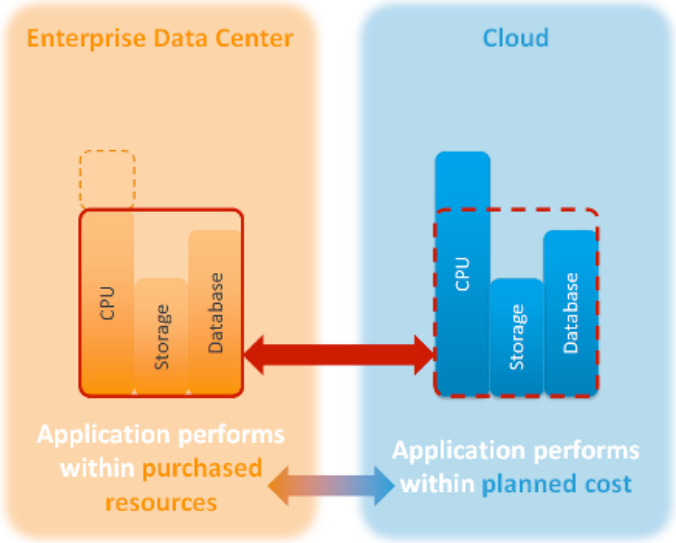
How cost evolves: Cost functions of our cloud resources, how our application consumes these resources, and how our users interact with our features

Let's assume our users search on average of 10 times (user interaction) before they purchase a product. One search transaction accesses—on average—the storage service 12 times (resource usage) which has a linear cost function. While the cost function will remain stable (as long as we don't change the cloud resource and the provider doesn't change the pricing), the resource usage can change easily from release to release because it's dependent on the search algorithm, bug fixes, caching algorithms, etc.—all usually decided by a single developer. For instance, optimizing the storage structure could lead to one additional storage request, which adds up if we have a couple of hundred thousand searches a month. Last, and the hardest to control, is the user interaction. Let's assume there's hype about a particular product and all your users want information and to read product reviews without the intention of buying. We want our application to smartly handle that situation without incurring immense cost.

TAKEAWAYS

It is obvious that we are dealing with a new cost structure in public cloud environments. Whereas we have fixed cost in the data center, we have transaction-based cost in public clouds. In other words, we have something like actual cost per user interaction. Financially, we have eliminated the risk of making unprofitable investments but now have introduced a chance of unpredictable costs.

Also, the findings above indicate a need for new responsibilities in our IT infrastructure. In the data center, we need to make sure that our applications are performing within our purchased resources, whereas in the cloud we need to have our applications performing within our planned costs.



The responsibility change of IT operations

SUMMARY AND OUTLOOK

Knowing that cost planning is a tricky matter, it might be the best idea to use pre-calculating only to get a rough estimate, and facilitate live-monitoring and fast release cycles to actually control cost. We strive to monitor the two dynamic factors of resource usage and user behavior on a transactional basis. Come back to this blog to learn how we can manage the new cost structure with a best-in-class application performance management and integrated user experience–management solution.

HOW EUE IMPACTS YOUR BOTTOM LINE

— Kristen Allmacher, April 25, 2012

Enterprise applications are the lifeblood of your business. These workhorse applications that allow companies to manage critical processes include ERP, CRM, SCM, PLM, and SFA.

These critical processes include the ability to process orders, manage financial systems, control inventory, track patients and medical records, schedule workflows, manage logistics, design new products, manage sales organizations, and support customers. Critical applications that are performing poorly can lead to lost business, decreased productivity, and inferior customer service.

Regardless of which class of application is in use, performance matters. The business impact of application performance and end-user experience (EUE) is the ultimate measure of an application's success.

The traditional reactive approach to the end-user experience is rapidly becoming obsolete in today's web-based and user-centric world. The new reality is that end-user expectations and high levels of performance against service-level agreements (SLAs) must be achieved or the organization risks losing business.

As a result, today's IT organizations must be able to answer accurately the following critical questions at any point in time:

- Are the business services the customers are relying on available?
- Are they providing acceptable and expected response time?
- Are there application-performance issues that are impacting users?

THE HIGH COST OF POOR PERFORMANCE

The costs associated with application downtime can be enormous. Most downtime costs average around $60,000 per hour, with 13% at more than $100,000/per hour.

When poor application performance leads to a lost loyal customer, the impact can be far-reaching. It's more than just the value of the purchase that is lost; it's the lifetime value of purchases those customers might have made.

The loss is not in profits alone, nor is its scope limited to certain businesses. Organizations of all sizes are seeing the impact of poor application performance. Consider that medium-size businesses (101 to 1,000 employees) are experiencing an average of nearly 140 hours of downtime every year and in turn losing an average of 1% of their annual revenue, or $867,000, to downtime.

Business Operation	Average Cost per Hour of Downtime
Communications: Converged Services	$10.0 million
Financial: Brokerage Operations	$6.45 million
Financial: Credit Card/Sales Authorization	$2.6 million
Media: Pay Per View	$150,000
Retail: Merchandise Sales	$140,000
Transportation: Airline Ticketing	$89,500
Media: Event Ticket Sales	$69,000

Source: Gartner, Dataquest, Contingency Planning Research and Others

Some costs are easier to quantify than others. Consider the "soft" costs of damaged brand, productivity, or customer satisfaction. Though not easily calculated, they can impact the business's bottom line.

THE MAJORITY OF APPLICATION PROBLEMS GO UNDETECTED

Industry reports vary on the exact percentage (some have it as high as 74%); however, a staggering number of end-user problems are undetected with current IT monitoring tools. In fact, the top three reported application-performance management challenges are as follows:

- Lack of visibility into the EUE
- The need for proactive problem detection before users call the help desk
- The ability to measure the business impact of poor application performance

In today's competitive marketplace, your customers, regardless of whether they're internal or external, expect perfect IT delivery of business services. A poor customer experience can cost the company a missed revenue opportunity and a potential future revenue stream. It stands to reason that by optimizing application performance, businesses will increase productivity, brand integrity, and business value.

FACT FINDERS: SORTING OUT THE TRUTH IN REAL USER MONITORING

— Klaus Enzenhofer, October 18, 2012

On my recent visits to Velocity, WebPerfDays, and Apps World in London, real user monitoring (RUM) was the hot topic. That triggered my thinking about the differences between vendors. They all promise the same for a varying range of prices—from free to a couple of thousand US dollars. I found out that there is a big difference and—depending on what you want to do with RUM—you want to make sure you understand the capabilities and limitations of the available solutions.

THE FALSE CLAIM OF 100% COVERAGE

All vendors claim to capture data from 100% of your users. When looking closer you see that many of these solutions—especially the "freemium" ones—rely on theW3C navigation timings. So my question is, How can I cover all users with W3C timings when these timings are not available on all browsers?

W3C timings are available only on new browsers. So what about IE6, IE7, IE8, the whole Safari browser family, and older Firefox and Chrome instances? Looking at current statistics they add up to 35% of the overall market share (**http://www.w3counter.com/globalstats.php**). The statements of vendors that rely on these timings to capture all users experience are simply not accurate.

THE PERFORMANCE IMPACT OF MONITORING

After finding that out I asked myself, "Are there any more deficiencies that can be found?"

I first thought about the collection mechanism, which reminded me of the challenges all the web analytics tools have. Data collection relies on the browser's onUnload event. The RUM tools have to collect the data till the last second of the lifecycle of the page and then send it off. Most SaaS solution vendors are using an image GET request to send the data to the collection instances. Modern browsers are optimizing this event because "Why should a browser download an image if the page is about to die?" Modern browsers like Chrome optimized this use case and simply do not execute the request at all or do not wait for a response if the data got sent. So again, I am losing data from my real end users. As a workaround, some vendors put a timeout in the onUnload event. I've seen timeouts with up to 500 ms, which impacts the next page that gets loaded. We want to improve the user experience/performance, but these tools are forcing the user to wait longer to move to the next page.

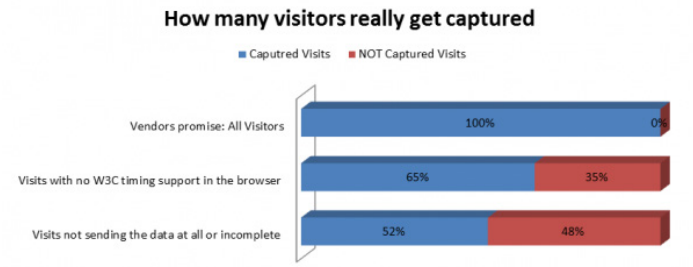
So we are losing all the old browsers and the modern ones that do not execute the data-collection requests. We are now far from 100% coverage.

DO THE MATH

Another argument you always hear is that the RUM solution allows you to find out more about the end-user environment's impact on page performance. The geographical region of the end user, the browser, the OS, or the device can result in slow page performance. But does this really work?

Let's do some simple math and figure out what this means to a page with 1 million visits a day:

- 1 million overall visits per day
- 1 million—35% visits with no W3C timing support in the browser
- 650,000—20% not sending the data correctly, completely, or at all
- 520,000 captured visits per day



Only 52% of visitors are captured by most RUM vendors due to limitations of browsers.

So we have reduced our base from 1 million to 520,000. Let's start with a breakdown into the different goupings:

- 520,000 broken down by 100 countries
- 520,000/100 = 5,200 visits/country/day
- 5,200 visits per country broken down by 20 browser versions
- 5,200/20 = 260 visits/country/browser version/day
- Let's break down the 260 visits further by 10 operating systems:

- 260/10 = 26 visits/country/browser version/operating system/day

We want to have the date on an hourly basis:

- 26/24 = ~1 visit/country/browser version/operating system/hour

The finding is that 1 million visits per day = ~ 1 visits/country/browser version/operating system/hour! We have done no sampling, we have only country-level data, and we are looking at visits and not page views!

To clarify: In this calculation I assume that the visits are evenly distributed over all countries but do not take into account that most solutions do sampling at a rate of 1–20% and look at visits with multiple page views instead of unique URIs—this seems to me a best-case scenario. In reality it can be even worse.

SO THEN WHY IS REAL USER MONITORING SO POPULAR?

Because it helps you to improve your users' experience! How can that work after knowing that we might not capture data from all our end users? You only have to change your expectations of what you want to achieve with real user monitoring.

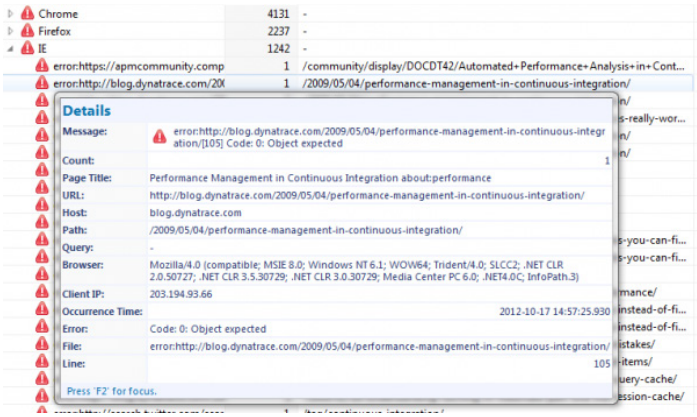
What you should expect from your RUM solution is

- Support for all browsers—not only the new browsers
- A reliable data-sending mechanism
- W3C timings support
- Functional health information such as errors from JavaScript and HTTP—not only timings
- AJAX/XHR-requests timing—not only timings for page loads
- The click path of a whole visit—not only separate page views
- Support for desktop browsers, mobile browsers, and mobile-native applications in combined view
- Landing- and exit-page analysis

If your selected solution provides all these features you can go an additional step and not only monitor your users, but also do real user-experience management (UEM). The following are some examples of what UEM allows you to do.

EXAMPLE 1: JAVASCRIPT ERRORS—WHICH ONE TO FIX FIRST?

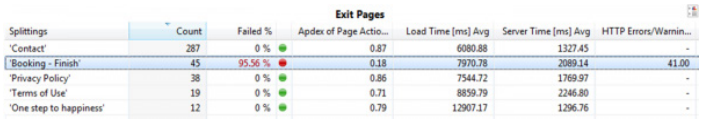
If your RUM- UEM solution provides you with JavaScript, errors you can start fixing problems right away. It should be able to show you which messages appear how often and in which browser, like in the screenshot below.



Detailed JavaScript error messages are captured for every visit and are easy accessible—grouped by browser, OS, or geographical location

EXAMPLE 2: WHY ARE MY CUSTOMERS LEAVING MY WEB SITE?

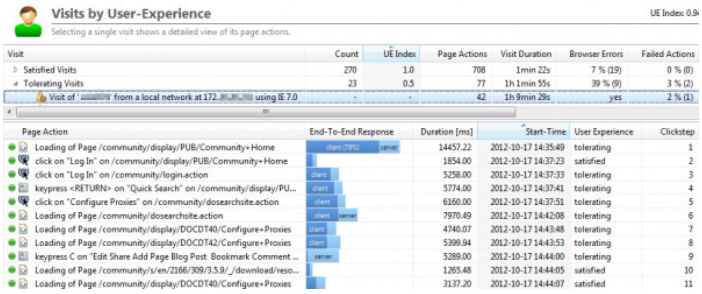
With the UEM you are now able to not only see that your customers are leaving your web site, but also to figure out if they had technical issues.



Looking at exit pages and correlating them with failure rate, performance, and user experience allows us to quickly identify why visitors leave the web site on these pages.

EXAMPLE 3: WHAT DID MY CUSTOMER DO ON THE APPLICATION BEFORE HE CALLED OUR SUPPORT CENTER?

Having every visit and all actions available makes it easy for the support center employees to look up the visit information as part of the triage process.



Seeing all actions the visitor really executed on the web site helps speed up the complaint-resolution process, as all facts are available.

EXAMPLE 4: CORRELATING PERFORMANCE TO BUSINESS

Analyzing the performance of every single visit and action not only allows us to pinpoint problems on individual pages, certain browsers, or geographical regions. It also allows us to correlate problems in the application to business. Knowing how much revenue is lost due to declined performance gives application owners better arguments when discussing investments in the infrastructure or additional R&D resources. The following dashboard correlates response time with the number of visitors by continent and the generated orders. Problems in the infrastructure that lead to performance problems of the application can then easily be correlated to lost revenue:

Correlating business values such as number of orders with page performance and infrastructure health opens a new line of communication between business and application owners.

CONCLUSION

W3C timings give us great insight but are available only in new browsers. Be aware of what your RUM solution vendor promises you and do not forget about the simple math. Set realistic expectations and look for solutions that support visits and health indicators like HTTP errors and JavaScript errors. Go real with the right expectations.

WHY PAGE SIZE MATTERS EVEN MORE FOR MOBILE WEB APPS

— Krzysztof Ziemianowicz, September 26, 2012

"Mobile" and "Web" sound like a perfect match, but reality often shows that these two trends form opposite forces. Recently we had an interesting engagement with one of our customers, in which we found how easy and dangerous it can be to inadvertently spoil end users' experience with a mobile web application by forgetting the practical limitations and consequences of web page size, which quickly grows with expansion of the application's functionality.

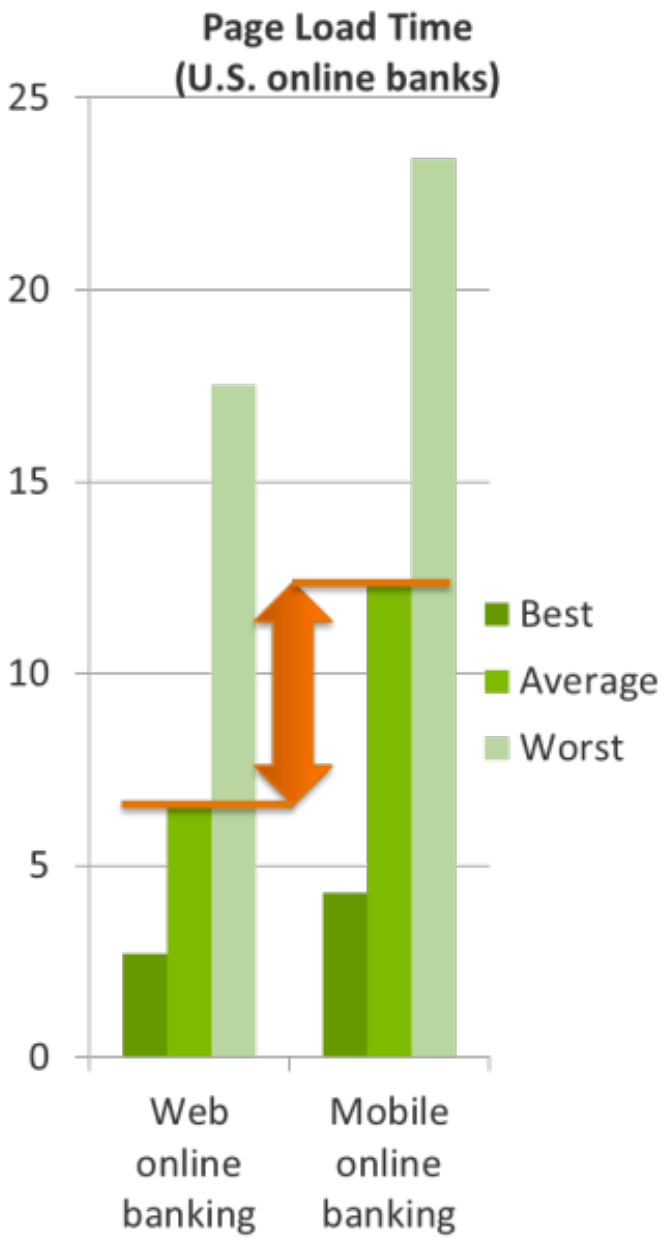
Last month one of our customers asked us to help diagnose why a mobile banking application that they rolled out was slow compared to the regular browser web application. The customer already knew that the mobile web application was slow, and business owners were disappointed by this known fact that was easy to verify experimentally by anyone having a smartphone and an account at the bank (and all of the bank's employees have it). During status meetings, typical explanations were discussed over and over again: mobile devices connect over slower networks, mobile devices are less capable than desktops and that is just how it works, and so on—with no conclusion and no action plan.

It is tempting to just assume that mobile applications have to be slow, especially since evidence can be found. Gomez Benchmarks, for example, published by Compuware (**http://www.gomez.com/us**-*banking-web-and-mobile-site-performance-indices/*), can be used to make a comparison like the one shown to the right, which shows that mobile-web banking applications are slower than desktop browser-based banking applications all around the world.

However, our customer has always challenged assumptions, including application-performance paradigms, and this time was no different. We agreed to have a look at this application with a planned proof of concept for extending the existing agentless end-user experience–monitoring into deep transaction-monitoring within the data center.

A 15-minute walk-through of the existing performance-overview reports for desktop and mobile web users revealed the root cause of the problem. First we confirmed that page-load time delivered to desktop and mobile web users differs by more than 100%, and we confirmed that data-center response time is not a problem:

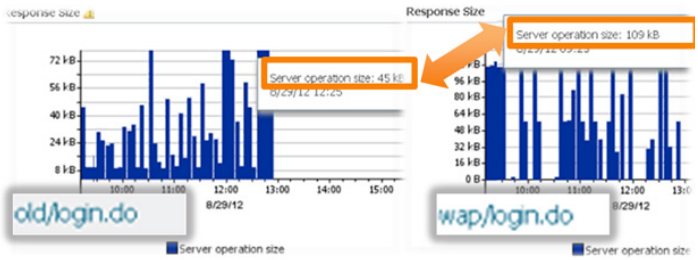
*The request for the mobile app (wap/**login.do**) took twice as long as for the desktop app (old/**login.do**), showing time spent on the network was the root cause of the problem.*



Load times of mobile-web banking apps are generally slower than desktop browser-based banking apps.

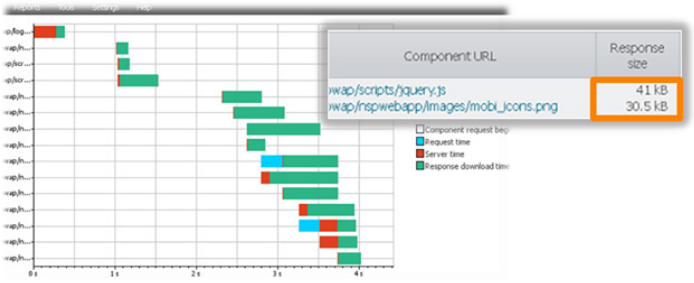
This simple illustration of server versus network time for the login page reveals how much time during page load is spent on generating the response within the data center and how much time is spent transferring the response to the client. The "old" is the desktop web application; the "wap" is the mobile web application. Clearly, mobile web–application users have to wait twice as long as desktop web users, and time is spent in response transfer over the network. We can also see how much time is spent by the client waiting for redirections before final page content is served, but this piece is out of scope of this analysis. Let's focus on the network time.

Network time is required to transfer data from server to client, and this time depends on two simple dimensions: **bandwidth offered by the network connection, and size of the page to transfer**. Naturally, these two dimensions have their own influencers, but let's leave this analysis for later. Let's start with checking the page sizes for desktop and mobile web applications.



The mobile web page is twice the size of the desktop version.

Here comes the surprise: **mobile web clients download more than twice as much data as desktop web clients**—109 KB versus 45 KB! This definitely was not expected, so we delved one step deeper to identify what exactly was transferred to the mobile web user that took so much bandwidth.

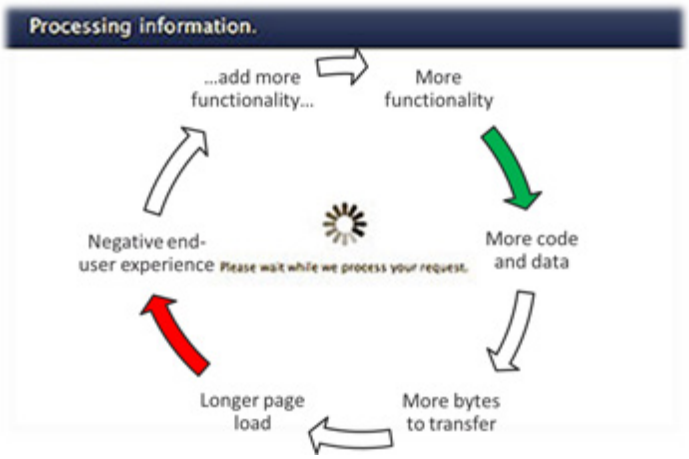


Large image and script files impact page-load performance on mobile web apps.

This waterfall chart illustrates a single mobile web page load for a single user—and it reveals what contributed to this page content. JavaScript and images form the bulk of that size, while even smart use of parallel TCP sessions to load content does not help; the truckload of data that has to be loaded to the mobile device takes time.

The waterfall chart also reveals **that the mobile client needs extensive time to run the JavaScript** and process the data—white spaces between horizontal bars indicate the time periods where client device is busy and does not open requests for more elements of the page. During these periods, the end user sees is the hourglass.

Designing mobile web applications requires a lot of creativity and effort, as mobile users tend to demand a sleeker experience with the application than desktop users. Mobile devices provide a narrower end-user error margin, and end users set the expectations bar high, basing them on their experience with state-of-the-art iOS and Android applications they use every day on the same device. This requirement is so important that it can easily overshadow the **old and simple relationship between page size and user wait time**. More functionality requires more code and data on the client side, which requires more bytes to be transferred to the client, which in turn means longer page-load times, and this negatively influences end-user experience.



Designing and Implementing mobile web apps easily leads to adding more code, thereby impacting page performance.

This is **a vicious circle of "sexy" web-application design**. Where is the balance between functionality and robustness? And overall, do you know the relationship between size, response time, and user satisfaction? And do you control it?

LESSONS LEARNED FROM REAL-WORLD BIG-DATA IMPLEMENTATIONS

— Michael Kopp, September 20, 2012

In the last weeks I visited several cloud and big-data conferences. Of special note, the Big Data Innovation in Boston gained me a lot of insight. Some people only consider the technology side of big-data technologies like Hadoop or Cassandra. However, business analysts discover big-data technologies as the means to leverage tons of existing data and ask questions about customer behavior and all sorts of relationships to drive business strategy. By doing that they are pushing their IT departments to run ever-bigger Hadoop environments and ever-faster real-time systems.

What's interesting from a technical side is that ad hoc analytics on existing data is allowed to take some time. However, ad hoc implies people waiting for an answer, meaning we are talking about minutes and not hours. Another interesting insight is that Hadoop environments are never static or standalone. Most companies take in new data on a continuous basis via technologies like Flume. This means Hadoop MapReduce jobs need to be able to keep up with the data flow, either by adding more hardware or by optimizing the existing hardware.

There are many drivers to big data, but the two most important are these: analytics and technical need for speed. Let's look at some of those and the resulting takeaways:

THE VALUE IS IN THE INSIGHT, NOT THE VOLUME

The value of big data is in the insights that the data can provide, not the sheer volume of it. The reason that more and more companies are keeping all of their log and transaction data is that they want to gain those insights. The sheer size of the data is an obstacle to this goal and has been for a long time. With big-data technologies this value can be harnessed.

DON'T FORGET THAT DATA ANALYSTS ARE PEOPLE TOO

Ad hoc analytics doesn't have to be instant, but must not take hours, either. It was interesting to see that time to result on ad hoc analytics is considered very important. This is because people are doing those queries, and people don't like to wait for hours. But even more important is that business analytics is often an iterative process. Ask a question, check the answer, refine or change the question. Hours-long MapReduce jobs are prohibitive to this process.

NEW DATA IS COMING IN ALL THE TIME

Big-data environments are constantly fed with new data. This is not major news, but I was still surprised by the constant reiteration of this fact. The constant data growth means that ad hoc queries either get slower over time or need to work on samples. To remedy this, companies are writing scrubbing and categorizing MapReduce jobs. These jobs basically strip out all the unimportant stuff and put cleansed, streamlined, easy-to-access data into new files. Instead of executing analytics against raw files, the analyst works on a cleansed data set. The implications are that scrubbing jobs need to be maintained all the time (as data input is changing over time) and they need to be able to keep up with the velocity of the input. MapReduce is not allowed to run for hours, but needs to be quick and iterative.

BIG DATA IS NOT CHEAP!

While it sounds obvious, it is something vendors don't talk about unless they're specifically asked. Hadoop requires a lot of hardware and a lot of expertise. The expertise is hard to come by as of yet. And while hardware might be cheap (you don't need expensive boxes for Hadoop), the bigger the environment the higher the operational cost. That operational cost is the reason some Hadoop vendors exist on services alone and also why customers are demanding better monitoring and management solutions.

DATA MUST BE ACCESSIBLE AT LOW LATENCIES TO PROVIDE VALUE

One very interesting fact is that most adopters that use Hadoop for analytics use it for ad hoc analytics and scrubbing, but not as a traditional warehouse. They use MapReduce to do the heavy lifting that is usually reserved for ETL jobs and put the resulting dimensions in existing data warehouses or into a NoSQL solution like HBase, Cassandra, or MongoDB. These solutions provide low-latency access semantics and are then integrated into the transactional application world—e.g., to provide recommendations to the end users.

This does not absolve them from optimizing their Hadoop environment where they can, but it gives them the much-needed real-time access that Hadoop so far does not provide. This also makes for additional complexity that needs to be maintained and monitored.

NOSQL SOLUTIONS NEED MANAGEMENT AND MONITORING, AS WELL

NoSQL solutions are most often used to provide low-latency databases with failover and horizontal scaling characteristics. As expected, practitioners quickly run into new issues like distribution and wrong access patterns. Most NoSQL solutions lack sophisticated monitoring or performance-analysis tools and require experts instead. Fortunately, several companies are working on providing those tools and some APM vendors work hard to support NoSQL databases similar to normal databases. This is emphasized by another interesting finding: with fast and scalable data storage, the application itself quickly becomes the response-time and scaling bottleneck.

APPLICATIONS USING NOSQL TECHNOLOGIES ARE MORE COMPLEX

Most NoSQL solutions surrender more-complex logic like joins in order to achieve horizontally scalable data distribution. That logic is moved to the application—arguably, this is where it should be anyway. NoSQL solutions require data to be stored in a query access–optimized way: de-normalization is the key. The flip side of storing data multiple times and the need to keep it in sync on updates is that the storage logic again becomes more complex. More application logic usually means less performance.

My conclusion as a performance engineer is relatively clear: big data requires performance-management and -monitoring tools to fulfill its promise in a cost-effective and timely manner. Here are some suggestions of what you should think about when you start a big data project:

- Large Hadoop environments are hard to manage and operate. Without automation in terms of deployment, operations, monitoring, and root-cause analysis, they quickly become unmanageable. Make sure to have a monitoring solution in place that informs you proactively of any infrastructure or software issues that would affect your operation. It needs to give you an easy way to pinpoint the root cause.
- The easiest way to identify new performance issues is to detect and analyze change. Adopt a life cycle and 24/7 production APM approach. It will enable you to notice changes in data and compute distribution over time. In addition a life-cycle approach will allow you to immediately pinpoint any negative changes introduced by a new software release.
- Don't just throw more and more hardware at the problem. Although you can use cheaper hardware for Hadoop, it's still cost. But more than that, you have to consider the operational drag. Every node you add will make traditional log-based analysis more complicated. Instead, ensure that you have an APM solution in place that lets you understand and optimize MapReduce jobs at their core and reduces both the time and resources it takes to run them.
- Your Hadoop cluster is no island, but will always be connected in some form or the other to a real time or at least transactional system. Make sure that you have a monitoring solution in place that can support both.
- NoSQL applications tend to have more-complex logic. The very performance and scalability of the store depends on correct data access and data distribution. A good monitoring solution allows you to monitor and optimize that additional complexity with ease; it also enables you to understand how your application accesses the data and how that access is distributed across your NoSQL cluster in your production system. The best way to ensure a scalable and fast NoSQL store is to ensure optimal distribution and access pattern.

CONCLUSION

Big data is still very much an emerging technology and its promises are huge. But in order to deliver on those promises it must be cost- and time-effective to those that harness its value —the business and not just technology experts.

WHY AVERAGES SUCK AND PERCENTILES ARE GREAT

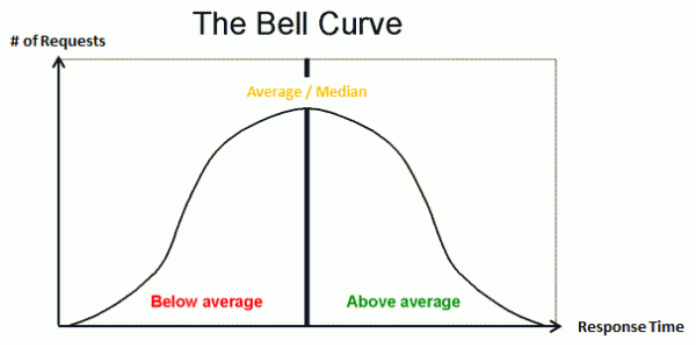
— Michael Kopp, November 14, 2012

Anyone that ever monitored or analyzed an application uses or has used averages. They are simple to understand and calculate. We tend to ignore just how wrong a picture averages paint of the world. To emphasis the point let me give you a real-world example outside of the performance space that I read recently in a newspaper.

The article was explaining that the average salary in a certain region in Europe was 1,900 Euros (to be clear, this would be quite good in that region!). However, a deeper look revealed that the majority, namely nine out of ten people, earned only around 1,000 Euros, and one out of ten earned 10,000. (I oversimplified this, of course, but you get the idea.) If you do the math you will see that the average of this is indeed 1900, but we can all agree that this does not represent the "average" salary as we would use the word in day-to-day life. So now let's apply this thinking to application performance.

THE AVERAGE RESPONSE TIME

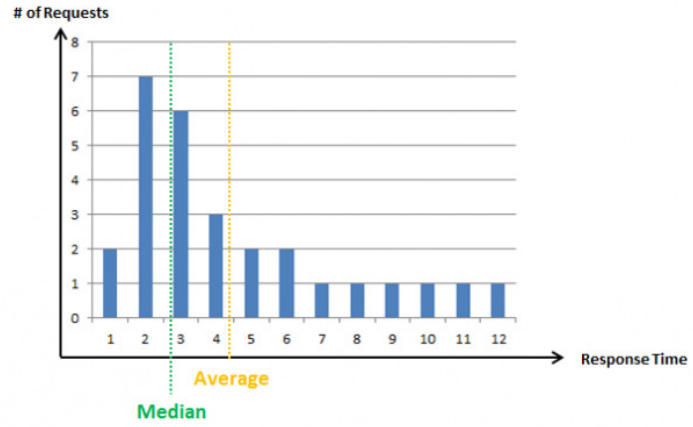
The average response time is by far the most commonly used metric in application performance management. We assume that this represents a "normal" transaction; however, this would be true only if the response time is always the same (all transactions run at equal speed) or the response-time distribution is roughly bell-shaped.



A bell curve represents the "normal" distribution of response times, in which the average and the median are the same. It rarely ever occurs in real applications.

In a bell curve the average (mean) and median are the same. In other words, observed performance would represent the majority (half or more than half) of the transactions.

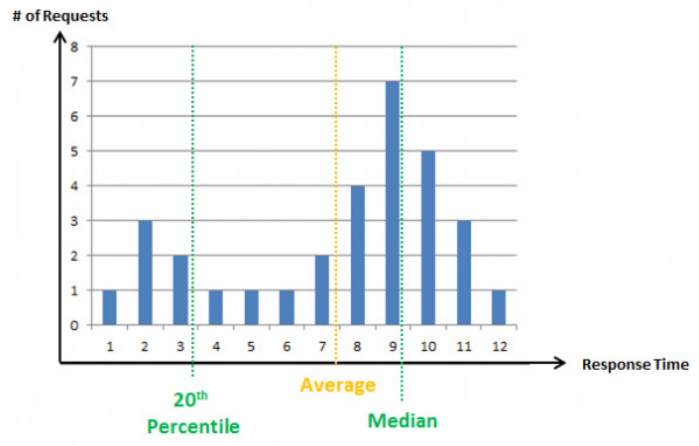
In reality most applications have few very heavy outliers; a statistician would say that the curve has a long tail. A long tail does not imply many slow transactions, but rather few that are magnitudes slower than the norm.



This is a typical response-time distribution with few but heavy outliers—it has a long tail. The average here is dragged to the right by the long tail.

We recognize that the average no longer represents the bulk of the transactions but can be a lot higher than the median.

You can argue that this is not a problem as long as the average doesn't look better than the median. I would disagree, but let's look at another real-world scenario experienced by many of our customers:



This is another typical response-time distribution. Here we have quite a few very fast transactions that drag the average to the left of the actual median.

In this case a considerable percentage (10–20%) of transactions are very, very fast, but the bulk of transactions are several times slower. The median would tell us the true story, but the average looks a lot faster than most of our transactions actually are. This is very typical in search engines or when caches are involved; some transactions are very fast, but the bulk are normal. Another reason for this scenario is failed transactions—more specifically, transactions that failed quickly. Many real-world applications have a failure rate of 1–10% (due to user errors or validation errors). These failed transactions are often magnitudes faster than the real ones and consequently distort an average.

Of course, performance analysts are not stupid and regularly try to compensate with higher-frequency charts (compensating by looking at smaller aggregates) and by taking in minimum and maximum observed response times. However, we can often do this only if we know the application very well. Those unfamiliar with the application might easily misinterpret the charts. Because of the depth and type of knowledge required for this, it's difficult to communicate your analysis to other people—think how many arguments between IT teams have been caused by this. And that's before we even begin to think about communicating with business stakeholders!

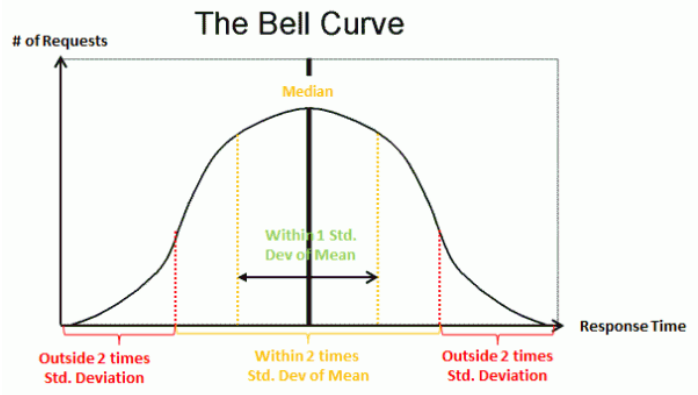
A better metric by far is percentiles, because they allow us to understand the distribution. But before we look at percentiles, let's take a look at a key feature in every production monitoring solution: automatic baselining and alerting.

AUTOMATIC BASELINING AND ALERTING

In real-world environments, performance gets attention when it is poor and has a negative impact on the business and users. But how can we identify performance issues quickly to prevent negative effects? We cannot alert on every slow transaction, since there are always some. In addition, most operations teams have to maintain a large number of applications are not familiar with all of them, so manually setting thresholds can be inaccurate, quite painful, and time-consuming.

The industry has come up with a solution called automatic baselining. Baselining calculates out the "normal" performance and alerts us only when an application slows down or produces more errors than usual. Most approaches rely on averages and standard deviations.

Without going into statistical details, this approach again assumes that the response times are distributed over a bell curve:

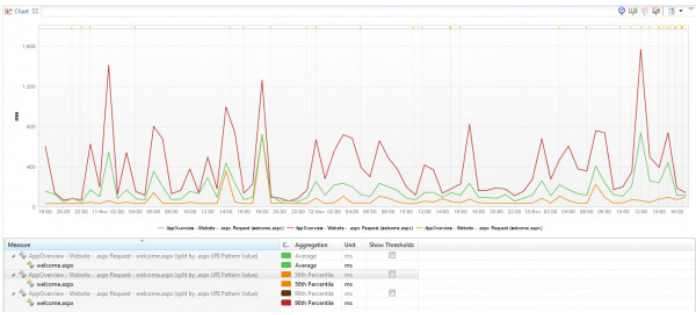


The standard deviation represents 33% of all transactions, with the mean as the middle. Two times standard deviation represents 66% and thus the majority; everything outside could be considered an outlier. However, most real-world scenarios are not bell-shaped.

Typically, transactions that are outside two times standard deviation are treated as slow and captured for analysis. An alert is raised if the average moves significantly. In a bell curve this would account for the slowest 16.5 percent (and you can, of course, adjust that); however, if the response-time distribution does not represent a bell curve it becomes inaccurate. We either end up with a lot of false positives (transactions that are a lot slower than the average but when looking at the curve lie within the norm) or we miss a lot of problems (due to false negatives). In addition, if the curve is not a bell then the average can differ a lot from the median. Applying a standard deviation to such an average can lead to quite a different result than you would expect! To work around this problem these algorithms have many tunable variables and a lot of "hacks" for specific use cases.

WHY I LOVE PERCENTILES

A percentile tells me which part of the curve I am looking at and how many transactions are represented by that metric. To visualize this, look at the following chart:



This chart shows the 50th and 90th percentile along with the average of the same transaction. It shows that the average is influenced heavily by the 90th—thus by outliers and not by the bulk of the transactions.

The green line represents the average. As you can see it is very volatile. The other two lines represent the 50th and 90th percentiles. As we can see, the 50th percentile (or median) is rather stable but has a couple of jumps. These jumps represent real performance degradation for the majority (50%) of the transactions. The 90th percentile (this is the start of the "tail") is a lot more volatile, which means that the outliers' slowness depends on data or user behavior. What's important here is that the average is heavily influenced (dragged) by the 90th percentile—the tail, rather than the bulk of the transactions.

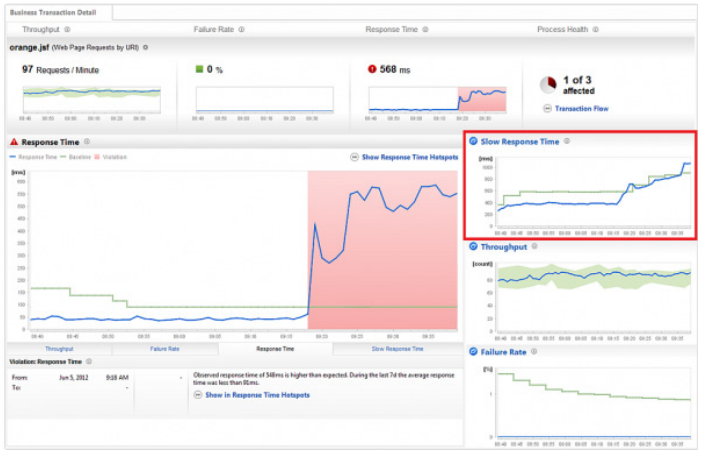
If the 50th percentile (median) of a response time is 500 ms that means that 50% of my transactions are either as fast as or faster than 500 ms. If the 90th percentile of the same transaction is at 1,000 ms it means that 90% are as fast or faster and only 10% are slower. The average in this case could either be lower than 500 ms (on a heavy front curve), a lot higher (long tail), or somewhere in between. A percentile gives me a much better sense of my real-world performance because it shows me a slice of my response-time curve.

For exactly that reason percentiles are perfect for automatic baselining. If the 50th percentile moves from 500 ms to 600 ms I know that 50% of my transactions suffered a 20% performance degradation. I would need to react to that.

In many cases we see that the 75th or 90th percentile does not change at all in such a scenario. This means the slow transactions didn't get any slower—only the normal ones did. Depending on how long your tail is, the average might not have moved at all in such a scenario!

In other cases we see the 98th percentile degrading from 1 second to 1.5 seconds while the 95th is stable at 900 ms. This means that your application as a whole is stable, but a few outliers got worse—nothing to worry about immediately. Percentile-based alerts do not suffer from false positives, are a lot less volatile, and don't miss any important performance degradations! Consequently, a baselining approach that uses percentiles does not require a lot of tuning variables to work effectively.

The screenshot below shows the median (50th percentile) for a particular transaction jumping from about 50 ms to about 500 ms and triggering an alert as it is significantly above the calculated baseline (green line). The chart labeled Slow Response Time, on the other hand, shows the 90th percentile for the same transaction. These "outliers" also show an increase in response time, but not significant enough to trigger an alert.



Here we see an automatic baselining dashboard with a violation at the 50th percentile. The violation is quite clear; at the same time, the 90th percentile (upper-right chart) does not violate. Because the outliers are so much slower than the bulk of the transactions, an average would have been influenced by them and would not have reacted quite as dramatically as the 50th percentile. We might have missed this clear violation!

HOW CAN WE USE PERCENTILES FOR TUNING?

Percentiles are also great for tuning, and giving your optimizations a particular goal. Let's say that something within my application is too slow in general and I need to make it faster. In this case I want to focus on bringing down the 90th percentile. This would ensure that the overall response time of the application goes down. In other cases I have unacceptably long outliers. I want to focus on bringing down response time for transactions beyond the 98th or 99th percentile (only outliers). We see a lot of applications that have perfectly acceptable performance for the 90th percentile, with the 98th percentile being magnitudes worse.

In throughput-oriented applications, on the other hand, I would want to make the majority of my transactions very fast, while accepting that an optimization makes a few outliers slower. I might therefore make sure that the 75th percentile goes down while trying to keep the 90th percentile stable or not getting a lot worse.

I could not make the same kind of observations with averages, minimum, and maximum, but with percentiles they are very easy indeed.

CONCLUSION

Averages are ineffective because they are too simplistic and one-dimensional. Percentiles are a really great and easy way of understanding the real performance characteristics of your application. They also provide a great basis for automatic baselining, behavioral learning, and optimization of your application with a proper focus. In short, percentiles are great!

TOP PERFORMANCE MISTAKES WHEN MOVING FROM TEST TO PRODUCTION: EXCESSIVE LOGGING

— Andreas Grabner, November 1, 2012

When things work great in testing it doesn't necessarily mean that everything works great in production. There are many things that might be different, such as much higher load than tested, different hardware, external services that couldn't be tested, and real users that execute transactions that weren't tested.

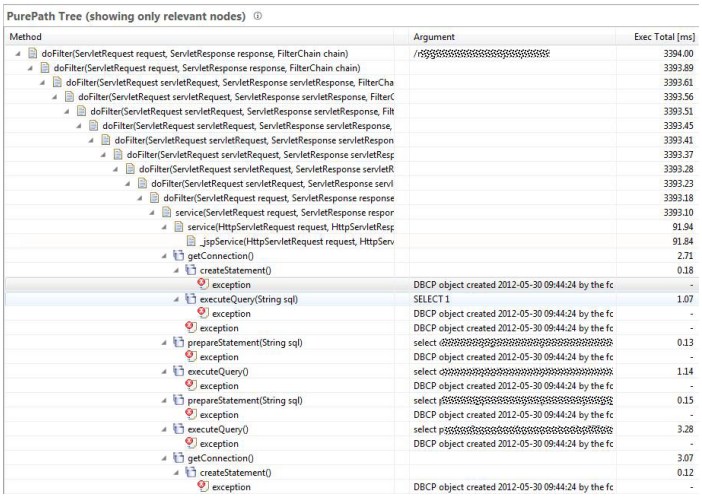
This blog post marks the start of a series covering the top performance mistakes when moving from testing to production. All examples are taken from real-life applications we have been working with. I will start the series with a topic that many application owners may not think about: excessive logging.

LOGGING IS IMPORTANT—BUT HANDLE WITH CARE

We all know that logging is important, as it provides valuable information when it comes to troubleshooting. Logging frameworks make it easy to generate log entries and, with the help of tools, it's becoming easier to consume and analyze log files when it is time to troubleshoot a problem. But what if there is too much log information that nobody looks at, that just fills up your file system and—worse—seriously impacts your application performance and thereby introduces a new problem to you application. The following two examples are taken from an online web application running on Tomcat. You can easily see how the current logging settings can become a real problem once this application gets deployed.

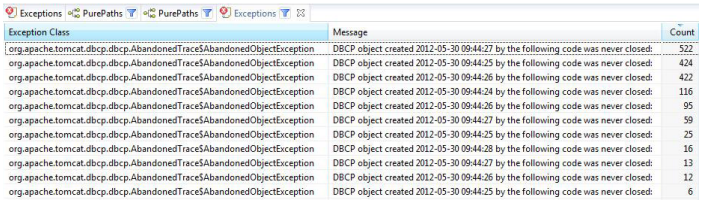
EXAMPLE #1: EXCESSIVE USE OF EXCEPTION OBJECTS TO LOG STACK TRACE

In this example the Tomcat connection pool was used with the setting "logAbandoned=true." Here the connection pool runs into a problem; it starts logging a lot of information, including full stack traces of where the application tried to get a connection from the pool. A typical transaction executed several hundred database queries. Most queries were executed on separate connections, which resulted in about as many getConnection calls as executedQuery calls.



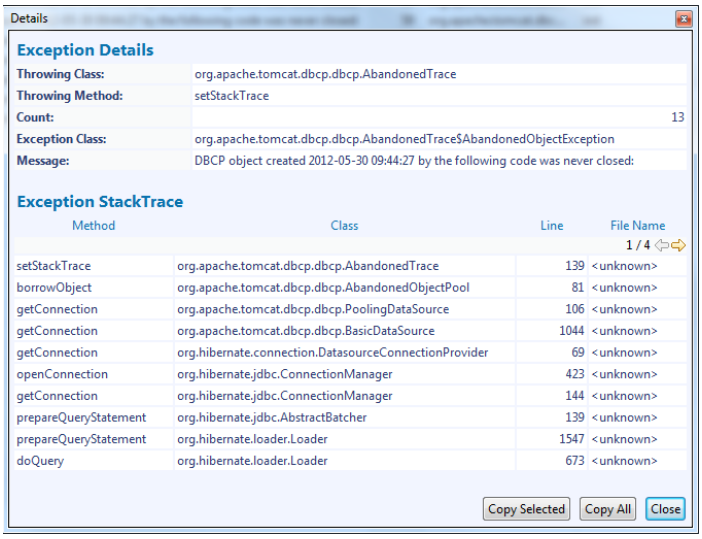
High overhead when creating hundreds of exception objects to get full stack-trace information

Looking just at the exception objects gives us a clearer picture of how much information gets collected but won't provide a whole lot of useful information.



Several hundred exception objects are created just for a single web request.

When looking at the exception details of these individual exception objects, it is easy to see which class actually creates these objects in order to obtain the stack-trace information for logging.



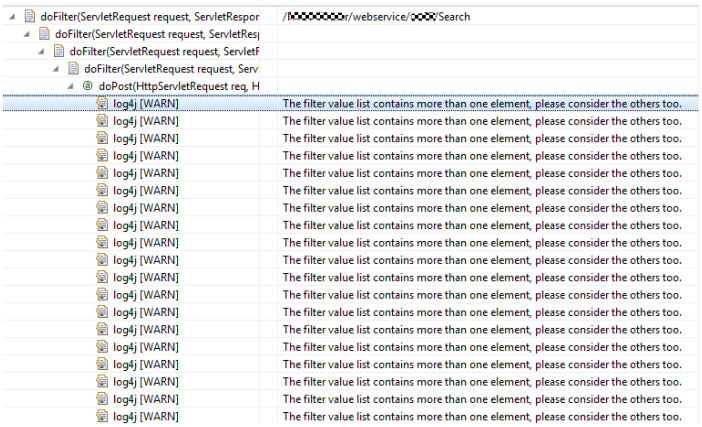
AbandonedTrace.setStackTrace *is the method that creates these exception objects.*

ADVICE FOR PRODUCTION

In the test environment this might not be a big issue if you're only simulating a fraction of the load expected on the live production system. It must, however, be a warning to the one responsible for moving this application to production. More information on this particular problem can be found here: **https://groups.google.com/forum/?fromgroups#!topic/mmbase-discuss/5x24EjBZMGA**.

EXAMPLE #2: TOO MUCH AND TOO GRANULAR USE OF LOGGING

Many applications have very detailed logging in order to make problem analysis easier in development and testing. When deploying an application in production people often forget to turn off DEBUG, FINE, FINEST, or INFO logs, which would flood the file system with useless log information and cause additional runtime overhead. It also often happens that log levels are used incorrectly, which leads to too many WARNING or ERROR logs for matters that are not a problem at all. This will make it very hard when analyzing log files to find the real problems. The following image shows a log message writing multiple times in the same transaction using the wrong log level.



Excessive logging and incorrect usage of severity level results in too much time spent analyzing log output.

ADVICE FOR PRODUCTION

- Make sure you configure your log levels correctly when deploying to production.
- Test these settings in your test environment and verify that there are no DEBUG, FINE, or other such log messages still logged out to the file system.
- Show the generated log files to the people that will have to use these log files in case of a production problem, and get their agreement that these log entries make sense and don't contain duplicated or "spam" messages.

ADDITIONAL READING

If you are interested in load testing, check out the post *To Load-Test or Not to Load-Test: That Is Not the Question*: **http://apmblog.compuware.com/2011/09/28/to-load-test-or-not-to-load-test-that-is-not-the-question/**

IT TAKES MORE THAN A TOOL! SWAROVSKI'S 10 REQUIREMENTS FOR CREATING AN APM CULTURE

— Andreas Grabner, November 21, 2012

Swarovski—the leading producer of cut crystal in the world—relies on its ecommerce store, much like other companies do in the highly competitive ecommerce environment. Swarovski's story is no different from others in this space: it started with "Let's build a web site to sell our products online" a couple of years ago and quickly progressed to "We sell to 60 million annual visitors across 23 countries in 6 languages." There were bumps along the road and they realized that it takes more than just a bunch of servers and tools to keep the site running.

WHY USE APM INSTEAD OF JUST A TOOL?

Swarovski relies on Intershop's ecommerce platform and faced several challenges as the ecommerce business grew rapidly. Those challenges required the company to apply application performance management (APM) practices to ensure it could fulfill the business requirements to keep pace with customer growth while maintaining an excellent user experience. The most insightful comment I heard was from René Neubacher, Senior eBusiness Technology Consultant at Swarovski: *"APM is not just about software. APM is a culture, a mindset, and a set of business processes. APM software supports that."*

René recently discussed Swarovski's journey to APM, what the initial problems were, and what requirements the company ended up having for APM and the tools needed to support Swarovski's APM strategy. The company reached the next level of maturity by establishing a Performance Center of Excellence that tackles application performance proactively throughout the organization instead of putting out fires reactively in production.

This blog post describes the challenges Swarovski faced, the questions that arose, and the new generation of APM requirements that paved the way forward.

THE CHALLENGE

Swarovski had traditional system monitoring in place on all its systems across the delivery chain, including web servers, application servers, SAP, database servers, external systems, and the network. Knowing that each individual component is up and running 99.99% of the time is great, but no longer sufficient.

How might individual component outages impact the user experience of online shoppers? *Who* is actually responsible for the end user experience and *how* should you monitor the complete delivery chain and not just the individual components? These and other questions came up when the ecommerce site attracted more customers, quickly followed by more complaints about the user experience:

APM includes getting a holistic view of the complete delivery chain and requires someone to be responsible for end-user experience.

QUESTIONS THAT NEEDED ANSWERS

In addition to *Who is responsible in case users complain?* the other questions that needed to be urgently addressed included the following:

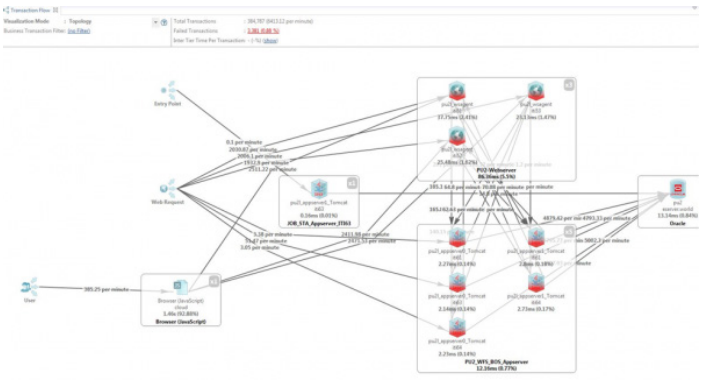
- How often is the service desk called before IT knows that there is a problem?
- How much time is spent in searching for system errors versus building new features?
- Do we have a process to find the root cause when a customer reports a problem?
- How do we visualize our services from the customer's point of view?
- How much revenue, brand image, and productivity are at risk or lost while IT is searching for the problem?
- What should be done when someone says "it's slow"?

THE 10 REQUIREMENTS

These questions triggered the need to move away from traditional system monitoring and develop the requirements for new-generation APM and user-experience management.

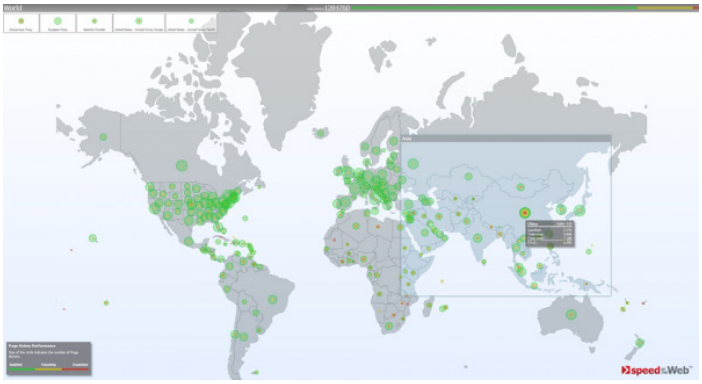
#1: SUPPORT STATE-OF-THE-ART ARCHITECTURE

Swarovski needed an approach that would work with its system architecture, now and in the future. The increase in interactive Web 2.0 and mobile applications had to be factored in to allow monitoring of end users from many different devices and regardless of whether they used a web application or mobile-native application as their access point.



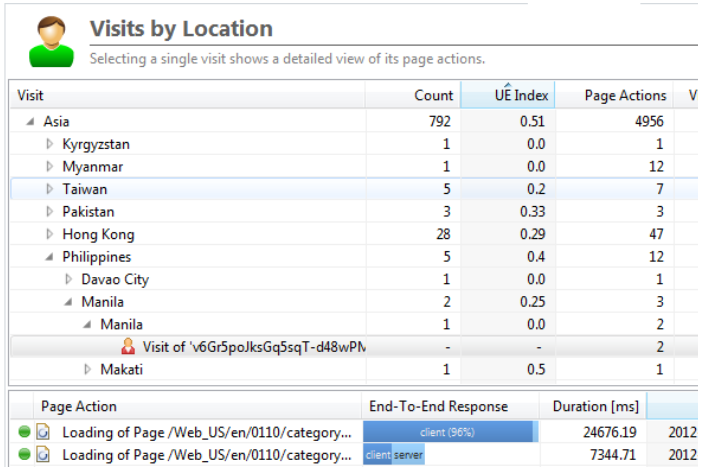
Transactions need to be followed from the browser all the way back to the database. It is important to support distributed transactions. This approach also helps to spot architectural and deployment problems immediately.

#2: 100% TRANSACTIONS AND CLICKS—NO AVERAGES



Measuring end-user performance of every customer interaction allows for quick identification of regional problems with CDNs, third parties, or latency.

Based on experience, Swarovski knew that looking at average values or sampled data would not be helpful when customers complained about bad performance. Responding to a customer complaint with "Our average user has no problem right now—sorry for your inconvenience" is not what you want your helpdesk engineers to do. Averages and sampling also hide the real problems in your system. Check out the blog post *Why Averages Suck*, **http://apm-blog.compuware.com/2012/11/14/why-averages-suck-and-percentiles-are-great/**, by Michael Kopp, for more detail.



Having 100% user interactions and transactions available makes it easy to identify the root cause of problems for individual users.

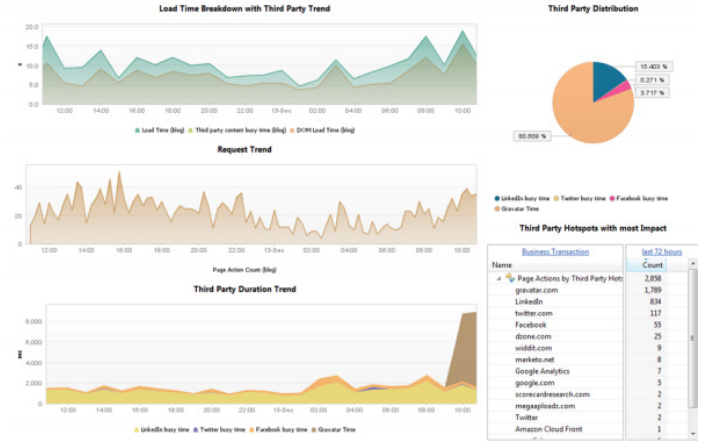
#3: BUSINESS VISIBILITY

As the business had a growing interest in the success of the ecommerce platform, IT had to demonstrate to the business what it took to fulfill the requirements and how business requirements are impacted by the availability or the lack of investment in the application delivery chain.

Correlating the number of visits with performance on incoming orders illustrates the measurable impact of performance on revenue and what it takes to support business requirements.

#4: IMPACT OF THIRD PARTIES AND CDNS

It was important to track not only transactions involving Swarovski's own data center, but *all* user interactions with the company's web site—even those delivered through content-delivery networks (CDNs) or third parties. All of these interactions make up the user experience and therefore *all* of them need to be analyzed.



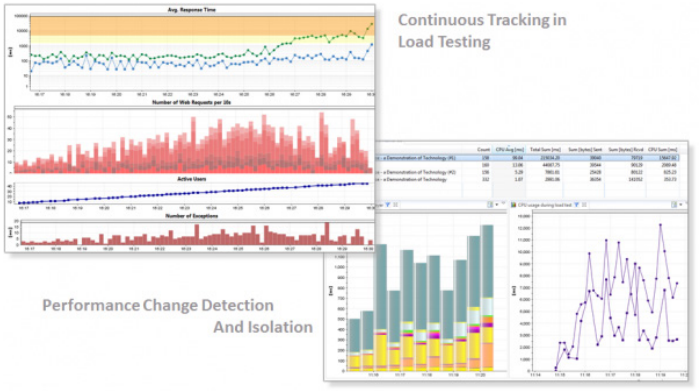
Seeing the actual load impact of third-party components or content delivered from CDNs enables IT to pinpoint user-experience problems that originate outside the company's own data center.

#5: ACROSS THE LIFECYCLE—SUPPORTING COLLABORATION AND TEARING DOWN SILOS

Continuously catching regressions in development by analyzing unit and performance tests allows application teams to become more proactive.

The APM initiative was started because Swarovski reacted to problems happening in production. Fixing these problems in production is only the first step. The ultimate goal is to become proactive by finding and fixing problems in development or testing—before they spill over into production.

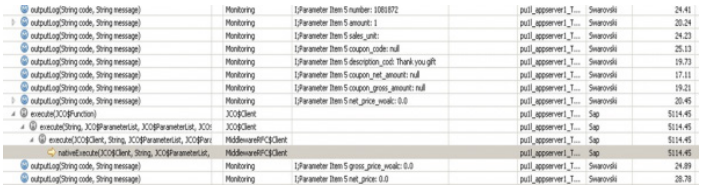
Instead of relying on different sets of tools with different capabilities, the requirement is to use one single solution that is designed to be used across the application lifecycle (developer workstation, continuous integration, testing, staging, and production). It will make it easier to share application performance data between lifecycle stages, allowing individuals to not only easily look at data from other stages, but also compare data to verify the impact and behavior of code changes between version updates.



Pinpointing integration and scalability issues, continuously, in acceptance and load testing makes testing more efficient and prevents problems from reaching production.

#6: DOWN TO THE SOURCE CODE

In order to speed up problem resolution, Swarovski's operations and development teams require as much code-level insight as possible—not only for the company's own engineers who are extending the Intershop ecommerce platform, but also for Intershop to improve its product. Knowing what part of the application code is not performing well with certain input parameters or under a specific load on the system eliminates tedious reproduction of the problem. The requirement is to lower the mean time to repair (MTTR) from as much as several days down to only a couple of hours.



The SAP connector turned out to have a performance problem. This method-level detailed information was captured without changing any code.

#7: ZERO/ACCEPTABLE OVERHEAD

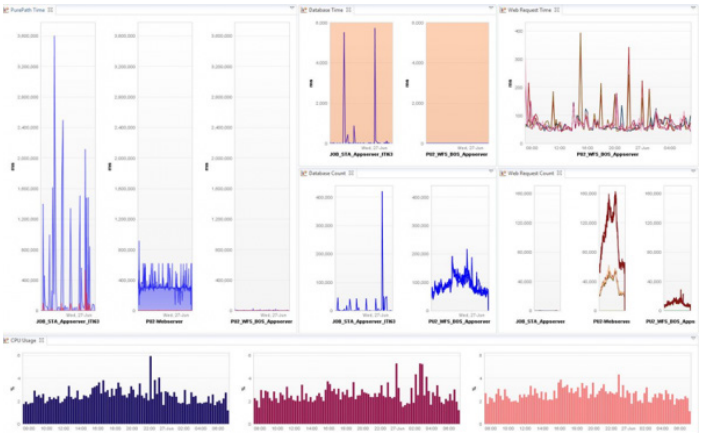
"Who are we kidding? There is nothing like zero overhead, especially when you need 100% coverage!" These were René Neubacher's words when he caught wind of that requirement. And he is right: once you start collecting information from a production system you add a certain amount of overhead. A better term than "zero overhead" would be "imperceptible overhead"—overhead that's so small, you don't notice it.

What is the exact number? It depends on your business and your users. The number should be worked out from the impact on the end-user experience rather than additional CPU, memory, or network bandwidth required in the data center. Swarovski knew it had to achieve less than 2% overhead on page load times in production, as anything more would have hurt its business.

#8: CENTRALIZED DATA COLLECTION AND ADMINISTRATION

Running a distributed ecommerce application that gets potentially extended to additional geographical locations requires an APM system with a centralized data-collection and administration option. It is not feasible to collect different types of performance information from different systems, servers, or even data centers. It would either require multiple analysis tools or data transformation to a single format to use it for proper analysis.

Instead of this approach, Swarovski needed a single unified APM system. Central administration was equally important, as the company needed to eliminate the reliance on remote IT administrators to make changes to the monitored system—for example, simple tasks such as changing the level of captured data or upgrading to a new version.



Storing and accessing performance data from a single, centralized repository enables fast and powerful analysis and visualization. For example, system metrics such as CPU utilization can be correlated with end-user response time or database execution time—all displayed on a single dashboard.

#9: AUTO-ADAPTING INSTRUMENTATION WITHOUT DIGGING THROUGH CODE

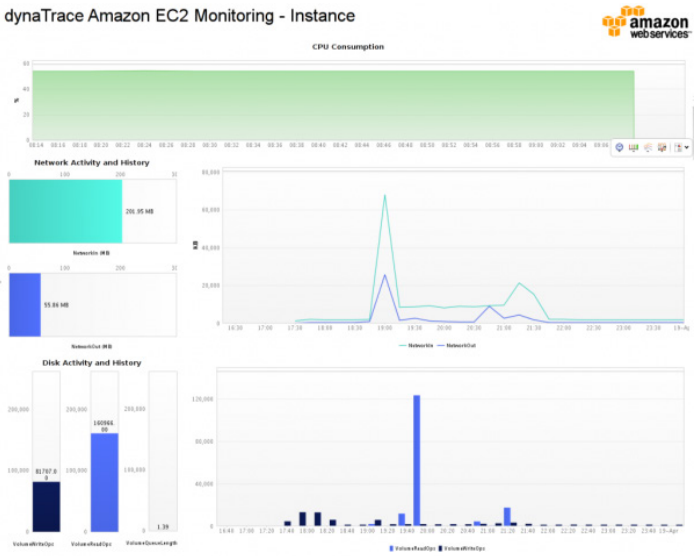
As the majority of Swarovski's application code is not developed in-house but provided by Intershop, it is mandatory to get insight into the application without doing any manual code changes. The APM system must auto-adapt to changes so that no manual configuration change is necessary when a new version of the application is deployed.

This means Swarovski can focus on making its applications contribute positively to business outcomes rather than spend time maintaining IT systems.

#10: ABILITY TO EXTEND

Swarovski's application is an always-growing and ever-changing IT environment. Elements that may have been deployed on physical boxes might be moved to virtualized environments or even into a public cloud environment.

Whatever the circumstance, the APM solution must be able to adapt to these changes and be extensible to consume new types of data sources—e.g., performance metrics from Amazon Cloud Services or VMware, Cassandra or other big-data solutions, or even extend to legacy mainframe applications and then bring these metrics into the centralized data repository and provide new insights into the application's performance.



Extending the application-monitoring capabilities to Amazon EC2, Microsoft Windows Azure, or another public or private cloud enables the analysis of the performance impact of these virtualized environments on end-user experience.

THE SOLUTION AND THE WAY FORWARD

Swarovski took the first step in implementing APM as a new process and mindset in the organization. The company is now in the next phase of implementing a Performance Center of Excellence. This allows movement from reactive performance troubleshooting to proactive performance prevention.

Stay tuned for more blog posts on the Performance Center of Excellence and how you can build one in your own organization. The key message is that it is not about just using a bunch of tools. It is about living and breathing performance throughout the organization. If you are interested in this topic, check out these blogs by Steve Wilson:

Proactive vs. Reactive: How to Prevent Problems Instead of Fixing Them Faster: **http://apmblog.compuware.com/2012/07/12/proactive-or-reactive-a-guide-to-prevent-problems-instead-of-fixing-them-faster/**

Performance in Development Is the Chief Cornerstone: **http://apmblog.compuware.com/2012/09/05/performance-in-development-is-the-chief-cornerstone/**

PERFORMANCE IS THE CORNERSTONE OF DEVELOPMENT

— Steve Wilson, September 5, 2012

In Roman architecture the stone a construction project was started from was called the cornerstone or foundation stone. It was the most important because this was the stone that the placement of all other stones would be based on. Great care was taken in making sure that the angles were correct. If there was a slight deviation, even one degree, it could cause structural issues for a foundation. When talking about performance across the lifecycle we must start at the foundation. In another **post** I talk about the need for looking at performance not just in a single phase but in all aspects of the lifecycle. In this article we will look at the need for performance to be the cornerstone of the development process.

WHAT IS THE CORNERSTONE FOR MOST DEVELOPMENT TEAMS?

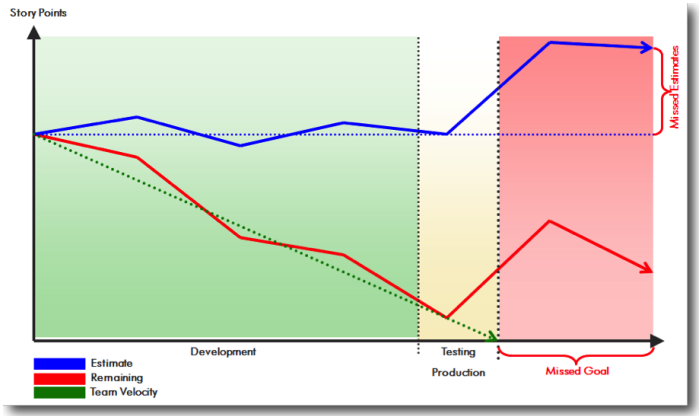
When researching what the top priorities for development teams are I was surprised to find that performance was rarely mentioned. Focus was typically on other issues—things like the scope of an application, limiting features for each build, or best coding practices. A recent Gartner survey showed that the top three priorities for development teams were as follows:

- Develop applications faster
- Expand the use of agile processes
- Reduce application-development costs

While performance was on the list it was down near the bottom. I find this ironic because in order to address the top three priorities performance should be the foundational priority. Let's look at each of the top three priorities from the survey in the context of having performance as the cornerstone.

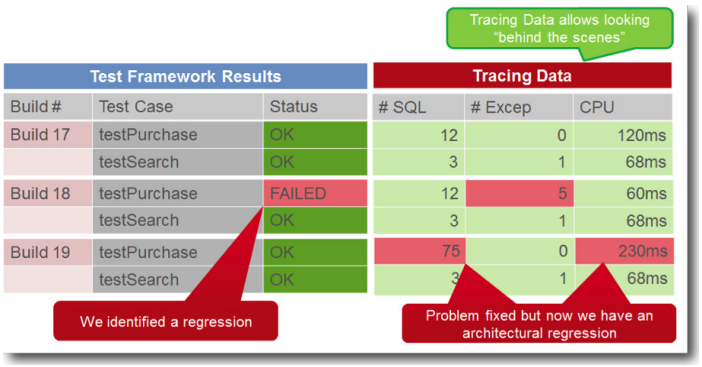
DEVELOP APPLICATIONS FASTER

With the advancement of web-delivered applications, companies are demanding increased interactions with customers and partners. The mobile revolution only compounds this. New features brought to market quickly can be the difference for a company. The agile process promotes quick iterations with small changes. This allows teams to churn out features rapidly. The drawback is that many times, in order to deliver applications on time, hard choices are made. Features are pushed and testing is seen as a luxury. As changes stack up problems that were introduced in previous sprints will become harder to unwind. This can come back to inhibit the scale of an application. A problem can be so embedded that the process to address it means tearing down several layers of functionality and rebuilding. Velocity stalls as teams must stop innovating to deal with bottlenecks that were introduced earlier in the development phase. The following graph visualizes what happens when performance is not treated as a high priority:



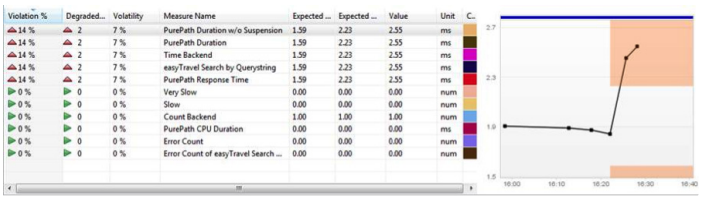
Performance must be a focus throughout development to avoid missed goals and missed expectations.

Let's look at how having performance as the foundation can impact faster releases. Using a performance platform earlier in the process gives developers insight into the performance of new application features. In addition to looking at test results from unit and integration tests, it is important to look at performance metrics that can be extracted from these test runs. This allows verification of functionality as well as architectural and performance problems by leveraging existing test assets. A best practice is to look at metrics such as *number of executed database statements*, *number of thrown exceptions*, *total execution time*, *created objects*, and *generated log statements*. The following table highlights some of these metrics to look at:



Analyzing performance Indicators for every test run identifies problems as they get introduced in the code.

Each code change is now tested for functional viability (*unit test status*), architectural correctness (*number of SQL statements*, *number of exceptions*, etc.) and performance (*execution time*, etc.). This indicates how the changes are impacting the overall quality and performance of an application. The following figure shows a different representation of this data, with built-in regression detection. The performance-management platform learns the expected value ranges for each performance indicator. If a metric is out of range, the platform automatically alerts on this regression:



Performance indicators that are out of the expected value range will automatically trigger alerts.

Developers see how the latest changes impact performance of a particular component that got tested. Each test can now generate a new task to address these automatically identified problems. The additional information that got captured in order to calculate these indicators helps when analyzing the actual regression. The following screenshot shows the comparison between the "last known good" and the problematic transaction that raised the alert.

Visual comparison makes it easy to identify the actual regression.

There is no need to write performance-specific user stories, as the platform automatically detects the test platform—in this case JUnit—and gives performance feedback for that build. Testing of performance becomes part of the automation process. With each build in the sprint performance is measured and compared to all pervious builds. It now just becomes a part of the sprint's overall "doneness."

An agile team can eliminate some of the stabilization sprints that are needed as performance defects have been vetted out in line. This keeps innovation moving and creates fewer chances for velocity to stall out due to stability issues.

EXPAND THE USE OF AGILE PROCESSES

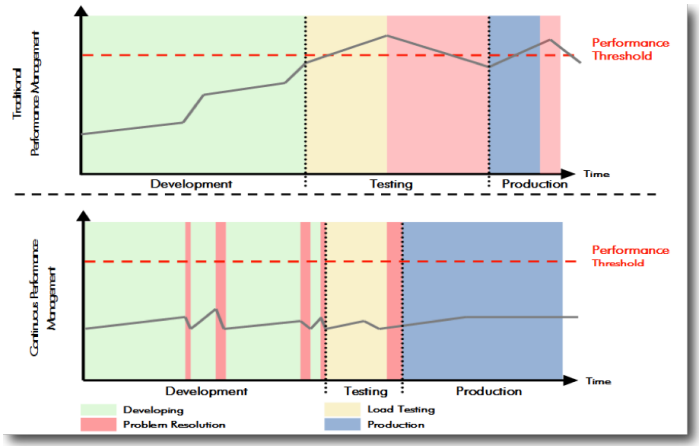
It is not a surprise that this is a top concern for development teams. More companies are moving from a waterfall structure to an agile process. The concept of test automation must be implemented for this transition to be successful. This is a time-consuming process, as most agile teams are not starting from scratch, but rather transitioning an application. Without these tests in place a core feature of the agile practice is lost. A main factor in regard to the adoption of agile practices and test automation is visibility. If no one is looking at the data from build to build then there is no reason to have the tests in the first place. Developers will focus on what is important to the completion of the project, and writing tests takes precious time and resources. This is a problem if you want to expand the use of agile practices.

With a solution that is integrated into the functional testing process, you have instance feedback about each test. Now the data provided is much richer than a simple pass/fail of tests. There is now an initial performance indication for each test. Teams are looking at this data for every build and visibility is high around it. There is a need to have coverage across the code base, as performance is now a critical component in the development stage. This need for understanding performance drives the adoption of test automation for agile teams, engraining it as standard practice.

REDUCE APPLICATION-DEVELOPMENT COSTS

This is just a reality of the world we live in now. For a development team time truly equals money. One developer costs x dollars per hour. In order to reduce the cost of development you must reduce the amount of time needed to deliver new features. This can be achieved by reducing the number of iterations needed to complete a new feature. Currently most performance is assessed at the testing level. As stated earlier, performance issues could be layered into several sprints that appear only in large-scale testing. This can set back a team several sprints, only adding to the cost.

Understanding performance earlier in the development cycle cuts down on the number of iterations needed to stabilize the code base for a new release. This allows for teams to keep the velocity of development very efficient and clean. The following charts illustrate this by comparing traditional performance management (performance as an after-thought) and continuous performance management (performance as a top priority).



Continuously focusing on performance results in better quality and on-time delivery.

Development is where ideas start to become reality. This is where applications specs begin to take shape and logical processes begin to morph into user interfaces and interactions. As you can see, the need to have performance in the development part of the lifecycle really is the cornerstone for an application. The slightest deviation can cause a ripple that may not appear until later in the lifecycle. So when we look at performance in this part of the lifecycle, it is surprising how little is done. In order to really become proactive, performance must be addressed from the start. The only way to do that is to have a platform in place that spans the lifecycle. In a future article I will move along the process and address how the acceptance team is crucial in managing performance and is a key component in becoming truly proactive.

If you are a dynaTrace user check out the following article on the Community Portal: *dynaTrace in Continuous Integration—The Big Picture*:
https://apmcommunity.compuware.com/community/display/PUB/dynaTrace+in+Continuous+Integration+-+The+Big+Picture

Compuware Corporation, the technology performance company, provides software, experts and best practices to ensure technology works well and delivers value. Compuware solutions make the world's most important technologies perform at their best for leading organizations worldwide, including 46 of the top 50 Fortune 500 companies and 12 of the top 20 most visited U.S. web sites. Learn more at: **compuware.com**.

Compuware Corporation World Headquarters • One Campus Martius • Detroit, MI 48226-5099

THE TECHNOLOGY
PERFORMANCE COMPANY